contiguous, into device address-space registers. Non-functional or impaired memory sections can be assigned a special address value which the system can interpret to avoid using that memory.

The second approach puts the burden of avoiding the bad devices on the system master or masters. CPUs and DMA controllers typically have some sort of translation look-aside buffers (TLBs) which map virtual to physical (bus) addresses. With relatively simple software, the TLBs can be programmed to use only working memory (data structures describing functional memories are easily generated). For masters which don't contain TLBs (for example, a video display generator), a small, simple RAM can be used to map a contiguous range of addresses onto the addresses of the functional memory devices.

Either scheme works and permits a system to have a significant percentage of non-functional devices and still continue to operate with the memory which remains. This means that systems built with this invention will have much improved reliability over existing systems, including the ability to build systems with almost no field failures.

Bus

The preferred bus architecture of this invention comprises 11 signals: BusData[0:7]; AddrValid; Clk1 and Clk2; plus an input reference level and power and ground lines connected in parallel to each device. Signals are driven onto

the bus during conventional bus cycles. The notation "Signal[i:j]" refers to a specific range of signals or lines, for example, BusData[0:7] means BusData0, BusData1, . . ., BusData7. The bus lines for BusData[0:7] signals form a byte-wide, multiplexed data/address/control bus. AddrValid is used to indicate when the bus is holding a valid address request, and instructs a slave to decode the bus data as an address and, if the address is included on that slave, to handle the pending request. The two clocks together provide a synchronized, high speed clock for all the devices on the bus. In addition to the bused signals, there is one other line (ResetIn, ResetOut) connecting each device in series for use during initialization to assign every device in the system a unique device ID number (described below in detail).

To facilitate the extremely high data rate of this external bus relative to the gate delays of the internal logic, the bus cycles are grouped into pairs of even/odd cycles. Note that all devices connected to a bus should preferably use the same even/odd labeling of bus cycles and preferably should begin operations on even cycles. This is enforced by the clocking scheme.

Protocol and Bus Operation

The bus uses a relatively simple, synchronous, split-transaction, block-oriented protocol for bus transactions. One

of the goals of the system is to keep the intelligence concentrated in the masters, thus keeping the slaves as simple as possible (since there are typically many more slaves than masters). To reduce the complexity of the slaves, a slave should preferably respond to a request in a specified time, sufficient to allow the slave to begin or possibly complete a device-internal phase including any internal actions that must precede the subsequent bus access phase. The time for this bus access phase is known to all devices on the bus - each master being responsible for making sure that the bus will be free when the bus access begins. Thus the slaves never worry about arbitrating for the bus. This approach eliminates arbitration in single master systems, and also makes the slave-bus interface simpler.

In a preferred implementation of the invention, to initiate a bus transfer over the bus, a master sends out a request packet, a contiguous series of bytes containing address and control information. It is preferable to use a request packet containing an even number of bytes and also preferable to start each packet on an even bus cycle.

The device-select function is handled using the bus data lines. AddrValid is driven, which instructs all slaves to decode the request packet address, determine whether they contain the requested address, and if they do, provide the data back to the master (in the case of a read request) or accept data from the master (in the case of a write requ st) in a data block

transfer. A master can also select a specific device by transmitting a device ID number in a request packet. In a preferred implementation, a special device ID number is chosen to indicate that the packet should be interpreted by all devices on the bus. This allows a master to broadcast a message, for example to set a selected control register of all devices with the same value.

The data block transfer occurs later at a time specified in the request packet control information, preferably beginning on an even cycle. A device begins a data block transfer almost immediately with a device-internal phase as the device initiates certain functions, such as setting up memory addressing, before the bus access phase begins. The time after which a data block is driven onto the bus lines is selected from values stored in slave access-time registers. The timing of data for reads and writes is preferably the same; the only difference is which device drives the bus. For reads, the slave drives the bus and the master latches the values from the bus. For writes the master drives the bus and the selected slave latches the values from the bus.

In a preferred implementation of this invention shown in Figure 4, a request packet 22 contains 6 bytes of data -- 4.5 address bytes and 1.5 control bytes. Each request packet uses all nine bits of the multiplexed data/address lines (AddrValid 23 + BusData[0:7] 24) for all six bytes of the request packet.

Setting 23 AddrValid = 1 in an otherwise unused ven cycle indicates the start of an request packet (control information). In a valid request packet, AddrValid 27 must be 0 in the last byte. Asserting this signal in the last byte invalidates the request packet. This is used for the collision detection and arbitration logic (described below). Bytes 25-26 contain the first 35 address bits, Address[0:35]. The last byte contains AddrValid 27 (the invalidation switch) and 28, the remaining address bits, Address[36:39], and BlockSize[0:3] (control information).

The first byte contains two 4 bit fields containing control information, AccessType[0:3], an op code (operation code) which, for example, specifies the type of access, and Master[0:3], a position reserved for the master sending the packet to include its master ID number. Only master numbers 1 through 15 are allowed - master number 0 is reserved for special system commands. Any packet with Master[0:3] = 0 is an invalid or special packet and is treated accordingly.

The AccessType field specifies whether the requested operation is a read or write and the type of access, for example, whether it is to the control registers or other parts of the device, such as memory. In a preferred implementation, AccessType[0] is a Read/Write switch: if it is a 1, then the operation calls for a read from the slave (the slav to r ad the requested memory block and drive the memory contents onto the

bus); if it is a 0, the operation calls for a write into the slave (the slave to read data from the bus and write it to memory). AccessType[1:3] provides up to 8 different access types for a slave. AccessType[1:2] preferably indicates the timing of the response, which is stored in an access-time register, AccessReg*N*. The choice of access-time register can be selected directly by having a certain op code select that register, or indirectly by having a slave respond to selected op codes with pre-selected access times (see table below). The remaining bit, AccessType[3] may be used to send additional information about the request to the slaves.

One special type of access is control register access, which involves addressing a selected register in a selected slave. In the preferred implementation of this invention, AccessType[1:3] equal to zero indicates a control register request and the address field of the packet indicates the desired control register. For example, the most significant two bytes can be the device ID number (specifying which slave is being addressed) and the least significant three bytes can specify a register address and may also represent or include data to be loaded into that control register. Control register accesses are used to initialize the access-time registers, so it is preferable to use a fixed response time which can be preprogrammed or even hard wired, for example the value in Acc ssReg0, preferably 8

cycles. Control register access can also be used to initialize or modify other registers, including address registers.

The method of this invention provides for access mode control specifically for the DRAMs. One such access mode determines whether the access is page mode or normal RAS access. In normal mode (in conventional DRAMS and in this invention), the DRAM column sense amps or latches have been precharged to a value intermediate between logical 0 and 1. This precharging allows access to a row in the RAM to begin as soon as the access request for either inputs (writes) or outputs (reads) is received and allows the column sense amps to sense data quickly. In page mode (both conventional and in this invention), the DRAM holds the data in the column sense amps or latches from the previous read or write operation. If a subsequent request to access data is directed to the same row, the DRAM does not need to wait for the data to be sensed (it has been sensed already) and access time for this data is much shorter than the normal access time. Page mode generally allows much faster access to data but to a smaller block of data (equal to the number of sense amps). However, if the requested data is not in the selected row, the access time is longer than the normal access time, since the request must wait for the RAM to precharge before the normal mode access can start. Two access-time registers in each DRAM preferably contain the access times to be used for normal and for page-mode accesses, respectively.

The access mode also determines whether the DRAM should precharge the sense amplifiers or should save the contents of the sense amps for a subsequent page mode access. Typical settings are "precharge after normal access" and "save after page mode access" but "precharge after page mode access" or "save after normal access" are allowed, selectable modes of operation. The DRAM can also be set to precharge the sense amps if they are not accessed for a selected period of time.

In page mode, the data stored in the DRAM sense amplifiers may be accessed within much less time than it takes to read out data in normal mode (~10-20 nS vs. 40-100 nS). This data may be kept available for long periods. However, if these sense amps (and hence bit lines) are not precharged after an access, a subsequent access to a different memory word (row) will suffer a precharge time penalty of about 40-100 nS because the sense amps must precharge before latching in a new value.

The contents of the sense amps thus may be held and used as a cache, allowing faster, repetitive access to small blocks of data. DRAM-based page-mode caches have been attempted in the prior art using conventional DRAM organizations but they are not very effective because several chips are required per computer word. Such a conventional page-mode cache contains many bits (for example, 32 chips x 4Kbits) but has very few independent storage entries. In other words, at any giv n point in time the s nse amps hold only a few different blocks or memory

"locales" (a single block of 4K words, in the example above). Simulations have shown that upwards of 100 blocks are required to achieve high hit rates (>90% of requests find the requested data already in cache memory) regardless of the size of each block. *See, for example*, Anant Agarwal, et. al., "An Analytic Cache Model," *ACM Transactions on Computer Systems*, Vol. 7(2), pp. 184-215 (May 1989).

The organization of memory in the present invention allows each DRAM to hold one or more (4 for 4MBit DRAMS) separately- addressed and independent blocks of data. A personal computer or workstation with 100 such DRAMs (i.e. 400 blocks or locales) can achieve extremely high, very repeatable hit rates (98-99% on average) as compared to the lower (50-80%), widely varying hit rates using DRAMS organized in the conventional fashion. Further, because of the time penalty associated with the deferred precharge on a "miss" of the page-mode cache, the conventional DRAM-based page-mode cache generally has been found to work less well than no cache at all.

For DRAM slave access, the access types are preferably used in the following way:

AccessType[1:3]	Use	AccessTime
0	Control Register Access	Fixed, 8[AccessReg0]
1	Unused	Fixed, 8[AccessReg0]
2-3	Unused	AccessReg1
4-5	Page Mode DRAM access	AccessReg2
6-7	Normal DRAM access	AccessReg3

Persons skilled in the art will recognize that a series of available bits could be designated as switches for controlling these access modes. For example:

AccessType[2] = page mode/normal switch

AccessType[3] = precharge/save-data switch

BlockSize[0:3] specifies the size of the data block transfer. If BlockSize[0] is 0, the remaining bits are the binary representation of the block size (0-7). If BlockSize[0] is 1, then the remaining bits give the block size as a binary power of 2, from 8 to 1024. A zero-length block can be interpreted as a special command, for example, to refresh a DRAM without returning any data, or to change the DRAM from page mode to normal access mode or vice-versa.

BlockSize[0:2]	Number of Byt s in Block
0-7	0-7 respectively
8	8
9	16
10	32
11	64
12	128
13	256
14	512
15	1024

Persons skilled in the art will recognize that other block size encoding schemes or values can be used.

In most cases, a slave will respond at the selected access time by reading or writing data from or to the bus over bus lines BusData[0:7] and AddrValid will be at logical 0. In a preferred embodiment, substantially each memory access will involve only a single memory device, that is, a single block will be read from or written to a single memory device.

Retry Format

In some cases, a slave may not be able to respond correctly to a request, e.g., for a read or write. In such a situation, the slave should return an error message, sometimes called a N(o)ACK(nowledge) or retry message. The retry message can include information about the condition requiring a retry, but this increases system requirements for circuitry in both slave and masters. A simple message indicating only that an error has occurred allows for a less complex slave, and the

master can take whatever action is needed to understand and correct the cause of the error.

For example, under certain conditions a slave might not be able to supply the requested data. During a page-mode access, the DRAM selected must be in page mode and the requested address must match the address of the data held in the sense amps or latches. Each DRAM can check for this match during a page-mode access. If no match is found, the DRAM begins precharging and returns a retry message to the master during the first cycle of the data block (the rest of the returned block is ignored). The master then must wait for the precharge time (which is set to accommodate the type of slave in question, stored in a special register, PreChargeReg), and then resend the request as a normal DRAM access (AccessType = 6 or 7).

In the preferred form of the present invention, a slave signals a retry by driving AddrValid true at the time the slave was supposed to begin reading or writing data. A master which expected to write to that slave must monitor AddrValid during the write and take corrective action if it detects a retry message. Figure 5 illustrates the format of a retry message 28 which is useful for read requests, consisting of 23 AddrValid=1 with Master[0:3] = 0 in the first (even) cycle. Note that AddrValid is normally 0 for data block transfers and that there is no master 0 (only 1 through 15 are allowed). All DRAMs and masters can easily recognize such a packet as an invalid request packet,

and therefore a retry message. In this type of bus transaction all of the fields except for Master[0:3] and AddrValid 23 may be used as information fields, although in the implementation described, the contents are undefined. Persons skilled in the art recognize that another method of signifying a retry message is to add a DataInvalid line and signal to the bus. This signal could be asserted in the case of a NACK.

Bus Arbitration

In the case of a single master, there are by definition no arbitration problems. The master sends request packets and keeps track of periods when the bus will be busy in response to that packet. The master can schedule multiple requests so that the corresponding data block transfers do not overlap.

The bus architecture of this invention is also useful in configurations with multiple masters. When two or more masters are on the same bus, each master must keep track of all the pending transactions, so each master knows when it can send a request packet and access the corresponding data block transfer. Situations will arise, however, where two or more masters send a request packet at about the same time and the multiple requests must be detected, then sorted out by some sort of bus arbitration.

There are many ways for each master to keep track of when the bus is and will be busy. A simple method is for each

master to maintain a bus-busy data structure, for example by maintaining two pointers, one to indicate the earliest point in the future when the bus will be busy and the other to indicate the earliest point in the future when the bus will be free, that is, the end of the latest pending data block transfer. Using this information, each master can determine whether and when there is enough time to send a request packet (as described above under Protocol) before the bus becomes busy with another data block transfer and whether the corresponding data block transfer will interfere with pending bus transactions. Thus each master must read every request packet and update its bus-busy data structure to maintain information about when the bus is and will be free.

With two or more masters on the bus, masters will occasionally transmit independent request packets during the same bus cycle. Those multiple requests will collide as each such master drives the bus simultaneously with different information, resulting in scrambled request information and neither desired data block transfer. In a preferred form of the invention, each device on the bus seeking to write a logical 1 on a BusData or AddrValid line drives that line with a current sufficient to sustain a voltage greater than or equal to the high-logic value for the system. Devices do not drive lines that should have a logical 0; those lines are simply held at a voltage corresponding to a low-logic value. Each master t sts the voltage on at least

some, preferably all, bus data and the AddrValid lines so the master can detect a logical '1' where the expected level is '0' on a line that it does not drive during a given bus cycle but another master does drive.

Another way to detect collisions is to select one or more bus lines for collision signalling. Each master sending a request drives that line or lines and monitors the selected lines for more than the normal drive current (or a logical value of ">1"), indicating requests by more than one master. Persons skilled in the art will recognize that this can be implemented with a protocol involving BusData and AddrValid lines or could be implemented using an additional bus line.

In the preferred form of this invention, each master detects collisions by monitoring lines which it does not drive to see if another master is driving those lines. Referring to Fig. 4, the first byte of the request packet includes the number of each master attempting to use the bus (Master[0:3]). If two masters send packet requests starting at the same point in time, the master numbers will be logical "or"ed together by at least those masters, and thus one or both of the masters, by monitoring the data on the bus and comparing what it sent, can detect a collision. For instance if requests by masters number 2 (0010) and 5 (0101) collide, the bus will be driven with the value Master[0:3]=7 (0010 + 0101 = 0111). Master number 5 will detect that the signal Master[2] = 1 and master 2 will detect that

Master[1] and Master[3] = 1, telling both masters that a collision has occurred. Another example is masters 2 and 11, for which the bus will be driven with the value Master[0:3]=11 (0010 + 1011 = 1011), and although master 11 can't readily detect this collision, master 2 can. When any collision is detected, each master detecting a collision drives the value of AddrValid 27 in byte 5 of the request packet 22 to 1, which is detected by all masters, including master 11 in the second example above, and forces a bus arbitration cycle, described below.

Another collision condition may arise where master A sends a request packet in cycle 0 and master B tries to send a request packet starting in cycle 2 of the first request packet, thereby overlapping the first request packet. This will occur from time to time because the bus operates at high speeds, thus the logic in a second-initiating master may not be fast enough to detect a request initiated by a first master in cycle 0 and to react fast enough by delaying its own request. Master B eventually notices that it wasn't supposed to try to send a request packet (and consequently almost surely destroyed the address that master A was trying to send), and, as in the example above of a simultaneous collision, drives a 1 on AddrValid during byte 5 of the first request packet 27 forcing an arbitration. The logic in the preferred implementation is fast enough that a master should detect a request packet by another master by cycle

3 of the first request packet, so no master is likely to attempt to send a potentially colliding request packet later than cycle 2.

Slave devices not need to detect a collision directly, but they must wait to do anything irrecoverable until the last byte (byte 5) is read to ensure that the packet is valid. A request packet with Master[0:3] equal to 0 (a retry signal) is ignored and does not cause a collision. The subsequent bytes of such a packet are ignored.

To begin arbitration after a collision, the masters wait a preselected number of cycles after the aborted request packet (4 cycles in a preferred implementation), then use the next free cycle to arbitrate for the bus (the next available even cycle in the preferred implementation). Each colliding master signals to all other colliding masters that it seeks to send a request packet, a priority is assigned to each of the colliding masters, then each master is allowed to make its request in the order of that priority.

Figure 6 illustrates one preferred way of implementing this arbitration. Each colliding master signals its intent to send a request packet by driving a single BusData line during a single bus cycle corresponding to its assigned master number (1-15 in the present example). During two-byte arbitration cycle 29, byte 0 is allocated to requests 1-7 from masters 1-7, respectively, (bit 0 is not used) and byte 1 is allocat d to

requests 8-15 from masters 8-15, respectively. At least one device and preferably each colliding master reads the values on the bus during the arbitration cycles to determine and store which masters desire to use the bus. Persons skilled in the art will recognize that a single byte can be allocated for arbitration requests if the system includes more bus lines than masters. More than 15 masters can be accommodated by using additional bus cycles.

A fixed priority scheme (preferably using the master numbers, selecting lowest numbers first) is then used to prioritize, then sequence the requests in a bus arbitration queue which is maintained by at least one device. These requests are queued by each master in the bus-busy data structure and no further requests are allowed until the bus arbitration queue is cleared. Persons skilled in the art will recognize that other priority schemes can be used, including assigning priority according to the physical location of each master.

System Configuration/Reset

In the bus-based system of this invention, a mechanism is provided to give each device on the bus a unique device identifier (device ID) after power-up or under other conditions as desired or needed by the system. A master can then use this device ID to access a specific device, particularly to s t or modify registers of the specified device, including the control

and address registers. In the preferred embodiment, one master is assigned to carry out the entire system configuration process. The master provides a series of unique device ID numbers for each unique device connected to the bus system. In the preferred embodiment, each device connected to the bus contains a special device-type register which specifies the type of device, for instance CPU, 4 MBit memory, 64 MBit memory or disk controller. The configuration master should check each device, determine the device type and set appropriate control registers, including access-time registers. The configuration master should check each memory device and set all appropriate memory address registers.

One means to set up unique device ID numbers is to have each device to select a device ID in sequence and store the value in an internal device ID register. For example, a master can pass sequential device ID numbers through shift registers in each of a series of devices, or pass a token from device to device whereby the device with the token reads in device ID information from another line or lines. In a preferred embodiment, device ID numbers are assigned to devices according to their physical relationship, for instance, their order along the bus.

In a preferred embodiment of this invention, the device ID setting is accomplished using a pair of pins on each device, ResetIn and R setOut. Th se pins handle normal logic signals and are used only during device ID configuration. On each rising

edge of the clock, each device copies ResetIn (an input) into a four-stage reset shift register. The output of the reset shift register is connected to ResetOut, which in turn connects to ResetIn for the next sequentially connected device. Substantially all devices on the bus are thereby daisy-chained together. A first reset signal, for example, while ResetIn at a device is a logical 1, or when a selected bit of the reset shift register goes from zero to non-zero, causes the device to hard reset, for example by clearing all internal registers and resetting all state machines. A second reset signal, for example, the falling edge of ResetIn combined with changeable values on the external bus, causes that device to latch the contents of the external bus into the internal device ID register (Device[0:7]).

To reset all devices on a bus, a master sets the R setIn line of the first device to a "1" for long enough to ensure that all devices on the bus have been reset (4 cycles times the number of devices -- note that the maximum number of devices on the preferred bus configuration is 256 (8 bits), so that 1024 cycles is always enough time to reset all devices.) Then ResetIn is dropped to "0" and the BusData lines are driven with the first followed by successive device ID numbers, changing after every 4 clock pulses. Successive devices set those device ID numbers into the corresponding device ID register as the falling edge of ResetIn propagates through the shift registers of

the daisy-chained devices. Figure 14 shows ResetIn at a first device going low while a master drives a first device ID onto the bus data lines BusData[0:3]. The first device then latches in that first device ID. After four clock cycles, the master changes BusData[0:3] to the next device ID number and ResetOut at the first device goes low, which pulls ResetIn for the next daisy-chained device low, allowing the next device to latch in the next device ID number from BusData[0:3]. In the preferred embodiment, one master is assigned device ID 0 and it is the r sponsibility of that master to control the ResetIn line and to drive successive device ID numbers onto the bus at the appropriate times. In the preferred embodiment, each device waits two clock cycles after ResetIn goes low before latching in a device ID number from BusData[0:3].

Persons skilled in the art recognize that longer device ID numbers could be distributed to devices by having each device read in multiple bytes from the bus and latch the values into the device ID register. Persons skilled in the art also recognize that there are alternative ways of getting device ID numbers to unique devices. For instance, a series of sequential numbers could be clocked along the ResetIn line and at a certain time each device could be instructed to latch the current reset shift register value into the device ID register.

The configuration master should choose and set an access time in each access-time regist r in each slave to a

period sufficiently long to allow the slave to perform an actual, desired memory access. For example, for a normal DRAM access, this time must be longer than the row address strobe (RAS) access time. If this condition is not met, the slave may not deliver the correct data. The value stored in a slave access-time register is preferably one-half the number of bus cycles for which the slave device should wait before using the bus in response to a request. Thus an access time value of '1' would indicate that the slave should not access the bus until at least two cycles after the last byte of the request packet has been received. The value of AccessReg0 is preferably fixed at 8 (cycles) to facilitate access to control registers.

The bus architecture of this invention can include more than one master device. The reset or initialization sequence should also include a determination of whether there are multiple masters on the bus, and if so to assign unique master ID numbers to each. Persons skilled in the art will recognize that there are many ways of doing this. For instance, the master could poll each device to determine what type of device it is, for example, by reading a special register then, for each master device, write the next available master ID number into a special register.

ECC

Error detection and correction ("ECC") methods well known in the art can be implemented in this system. ECC

information typically is calculated for a block of data at the time that block of data is first written into memory. The data block usually has an integral binary size, e.g. 256 bits, and the ECC information uses significantly fewer bits. A potential problem arises in that each binary data block in prior art schemes typically is stored with the ECC bits appended, resulting in a block size that is not an integral binary power.

In a preferred embodiment of this invention, ECC information is stored separately from the corresponding data, which can then be stored in blocks having integral binary size. ECC information and corresponding data can be stored, for example, in separate DRAM devices. Data can be read without ECC using a single request packet, but to write or read error-corrected data requires two request packets, one for the data and a second for the corresponding ECC information. ECC information may not always be stored permanently and in some situations the ECC information may be available without sending a request packet or without a bus data block transfer.

In a preferred embodiment, a standard data block size can be selected for use with ECC, and the ECC method will determine the required number of bits of information in a corresponding ECC block. RAMs containing ECC information can be programmed to store an access time that is equal to: (1) the access time of the normal RAM (containing data) plus the time to acc ss a standard data block (for corrected data) minus the time

to send a request packet (6 bytes); or (2) the access time of a normal RAM minus the time to access a standard ECC block minus the time to send a request packet. To read a data block and the corresponding ECC block, the master simply issues a request for the data immediately followed by a request for the ECC block. The ECC RAM will wait for the selected access time then drive its data onto the bus right after (in case (1) above)) the data RAM has finished driving out the data block. Persons skilled in the art will recognize that the access time described in case (2) above can be used to drive ECC data before the data is driven onto the bus lines and will recognize that writing data can be done by analogy with the method described for a read. Persons skilled in the art will also recognize the adjustments that must be made in the bus-busy structure and the request packet arbitration methods of this invention in order to accommodate these paired ECC requests.

Since this system is quite flexible, the system designer can choose the size of the data blocks and the number of ECC bits using the memory devices of this invention. Note that the data stream on the bus can be interpreted in various ways. For instance the sequence can be 2^n data bytes followed by 2^m ECC bytes (or vice versa), or the sequence can be 2^k iterations of 8 data bytes plus 1 ECC byte. Other information, such as information used by a directory-based cache coherence scheme, can also be managed this way. *See*, for example, Anant Agarwal, et

al., "Scaleable Directory Schemes for Cache Consistency," 15th *International Symposium on Computer Architecture*, June 1988, pp. 280-289. Those skilled in the art will recognize alternative methods of implementing ECC schemes that are within the teachings of this invention.

Low Power 3-D Packaging

Another major advantage of this invention is that it drastically reduces the memory system power consumption. Nearly all the power consumed by a prior art DRAM is dissipated in performing row access. By using a single row access in a single RAM to supply all the bits for a block request (compared to a row-access in each of multiple RAMs in conventional memory systems) the power per bit can be made very small. Since the power dissipated by memory devices using this invention is significantly reduced, the devices potentially can be placed much closer together than with conventional designs.

The bus architecture of this invention makes possible an innovative 3-D packaging technology. By using a narrow, multiplexed (time-shared) bus, the pin count for an arbitrarily large memory device can be kept quite small - on the order of 20 pins. Moreover, this pin count can be kept constant from one generation of DRAM density to the next. The low power dissipation allows each package to be smaller, with narrower pin pitches (spacing between the IC pins). With current surface

mount technology supporting pin pitches as low as 20 mils, all off-device connections can be implemented on a single edge of the memory device. Semiconductor die useful in this invention preferably have connections or pads along one edge of the die which can then be wired or otherwise connected to the package pins with wires having similar lengths. This geometry also allows for very short leads, preferably with an effective lead length of less than 4 mm. Furthermore, this invention uses only bused interconnections, i.e., each pad on each device is connected by the bus to the corresponding pad of each other device.

The use of a low pin count and an edge-connected bus permits a simple 3-D package, whereby the devices are stacked and the bus is connected along a single edge of the stack. The fact that all of the signals are bused is important for the implementation of a simple 3-D structure. Without this, the complexity of the "backplane" would be too difficult to make cost effectively with current technology. The individual devices in a stack of the present invention can be packed quite tightly because of the low power dissipated by the entire memory system, permitting the devices to be stacked bumper-to-bumper or top to bottom. Conventional plastic-injection molded small outline (SO) packages can be used with a pitch of about 2.5 mm (100 mils), but the ultimate limit would be the device die thickn ss, which is

about an order of magnitude smaller, 0.2-0.5 mm using current wafer technology.

Bus Electrical Description

By using devices with very low power dissipation and close physical packing, the bus can be made quite short, which in turn allows for short propagation times and high data rates. The bus of a preferred embodiment of the present invention consists of a set of resistor-terminated controlled impedance transmission lines which can operate up to a data rate of 500 MHz (2 ns cycles). The characteristics of the transmission lines are strongly affected by the loading caused by the DRAMs (or other slaves) mounted on the bus. These devices add lumped capacitance to the lines which both lowers the impedance of the lines and decreases the transmission speed. In the loaded environment, the bus impedance is likely to be on the order of 25 ohms and the propagation velocity about $c/4$ (c = the speed of light) or 7.5 cm/ns. To operate at a 2 ns data rate, the transit time on the bus should preferably be kept under 1 ns, to leave 1 ns for the setup and hold time of the input receivers (described below) plus clock skew. Thus the bus lines must be kept quite short, under about 8 cm for maximum performance. Lower performance systems may have much longer lines, e.g. a 4 ns bus may have 24 cm lines (3 ns transit time, 1 ns setup and hold time).

In the pref rred embodiment, the bus uses current source drivers. Each output must be able to sink 50 mA, which provides an output swing of about 500 mV or more. In the preferred embodiment of this invention, the bus is active low. The unasserted state (the high value) is preferably considered a logical zero, and the asserted value (low state) is therefore a logical 1. Those skilled in the art understand that the method of this invention can also be implemented using the opposite logical relation to voltage. The value of the unasserted state is set by the voltage on the termination resistors, and should be high enough to allow the outputs to act as current sources, while being as low as possible to reduce power dissipation. These constraints may yield a termination voltage about 2V above ground in the preferred implementation. Current source drivers cause the output voltage to be proportional to the sum of the sources driving the bus.

Referring to Fig.7, although there is no stable condition where two devices drive the bus at the same time, conditions can arise because of propagation delay on the wires where one device, A 41, can start driving its part of the bus 44 while the bus is still being driven by another device, B 42 (already asserting a logical 1 on the bus). In a system using current drivers, when B 42 is driving the bus (before time 46), the valu at points 44 and 45 is logical 1. If B 42 switches off at time 46 just when A 41 switches on, the additional driv by

device A 41 causes the voltage at the output 44 of A 41 to drop briefly below the normal value. The voltage returns to its normal value at time 47 when the effect of device B 42 turning off is felt. The voltage at point 45 goes to logical 0 when device B 42 turns off, then drops at time 47 when the effect of device A 41 turning on is felt. Since the logical 1 driven by current from device A 41 is propagated irrespective of the previous value on the bus, the value on the bus is guaranteed to settle after one time of flight (t_f) delay, that is, the time it takes a signal to propagate from one end of the bus to the other. If a voltage drive was used (as in ECL wired-ORing), a logical 1 on the bus (from device B 42 being previously driven) would prevent the transition put out by device A 41 being felt at the most remote part of the system, e.g., device 43, until the turnoff waveform from device B 42 reached device A 41 plus one time of flight delay, giving a worst case settling time of twice the time of flight delay.

Clocking

Clocking a high speed bus accurately without introducing error due to propagation delays can be implemented by having each device monitor two bus clock signals and then derive internally a device clock, the true system clock. The bus clock information can be sent on one or two lines to provide a mechanism for each bused device to generat an internal device

clock with zero skew relative to all the other device clocks. Referring to Figure 8, in the preferred implementation, a bus clock generator 50 at one end of the bus propagates an early bus clock signal in one direction along the bus, for example on line 53 from left to right, to the far end of the bus. The same clock signal then is passed through the direct connection shown to a second line 54, and returns as a late bus clock signal along the bus from the far end to the origin, propagating from right to left. A single bus clock line can be used if it is left unterminated at the far end of the bus, allowing the early bus clock signal to reflect back along the same line as a late bus clock signal.

Figure 8b illustrates how each device 51, 52 receives each of the two bus clock signals at a different time (because of propagation delay along the wires), with constant midpoint in time between the two bus clocks along the bus. At each device 51, 52, the rising edge 55 of Clock1 53 is followed by the rising dge 56 of Clock2 54. Similarly, the falling edge 57 of Clock1 53 is followed by the falling edge 58 of Clock2 54. This waveform relationship is observed at all other devices along the bus. Devices which are closer to the clock generator have a greater separation between Clock1 and Clock2 relative to devices farther from the generator because of the longer time required for each clock pulse to traverse the bus and return along line 54, but the midpoint in time 59, 60 b tween corresponding rising

or falling edges is fixed because, for any given device, the length of each clock line between the far end of the bus and that device is equal. Each device must sample the two bus clocks and generate its own internal device clock at the midpoint of the two.

Clock distribution problems can be further reduced by using a bus clock and device clock rate equal to the bus cycle data rate divided by two, that is, the bus clock period is twice the bus cycle period. Thus a 500 MHz bus preferably uses a 250 MHz clock rate. This reduction in frequency provides two benefits. First it makes all signals on the bus have the same worst case data rates -- data on a 500 MHz bus can only change every 2 ns. Second, clocking at half the bus cycle data rate makes the labeling of the odd and even bus cycles trivial, for example, by defining even cycles to be those when the internal d vice clock is 0 and odd cycles when the internal device clock is 1.

Multiple Buses

The limitation on bus length described above restricts the total number of devices that can be placed on a single bus. Using 2.5 mm spacing between devices, a single 8 cm bus will hold about 32 devices. Persons skilled in the art will recognize certain applications of the present invention wherein the overall data rate on the bus is adequate but memory or processing

requirements necessitate a much larger number of devices (many more than 32). Larger systems can easily be built using the teachings of this invention by using one or more memory subsystems, designated primary bus units, each of which consists of two or more devices, typically 32 or close to the maximum allowed by bus design requirements, connected to a transceiver device.

Referring to Figure 9, each primary bus unit can be mounted on a single circuit board 66, sometimes called a memory stick. Each transceiver device 19 in turn connects to a transceiver bus 65, similar or identical in electrical and other respects to the primary bus 18 described at length above. In a preferred implementation, all masters are situated on the transceiver bus so there are no transceiver delays between masters and all memory devices are on primary bus units so that all memory accesses experience an equivalent transceiver delay, but persons skilled in the art will recognize how to implement systems which have masters on more than one bus unit and memory devices on the transceiver bus as well as on primary bus units. In general, each teaching of this invention which refers to a memory device can be practiced using a transceiver device and one or more memory devices on an attached primay bus unit. Other devices, generically referred to as peripheral devices, including disk controllers, video controllers or I/O devices can also be attached to either the transceiver bus or a primary bus unit, as

desired. Persons skilled in the art will recognize how to use a single primary bus unit or multiple primary bus units as needed with a transceiver bus in certain system designs.

The transceivers are quite simple in function. They detect request packets on the transceiver bus and transmit them to their primary bus unit. If the request packet calls for a write to a device on a transceiver's primary bus unit, that transceiver keeps track of the access time and block size and forwards all data from the transceiver bus to the primary bus unit during that time. The transceivers also watch their primary bus unit, forwarding any data that occurs there to the transceiver bus. The high speed of the buses means that the transceivers will need to be pipelined, and will require an additional one or two cycle delay for data to pass through the transceiver in either direction. Access times stored in masters on the transceiver bus must be increased to account for transceiver delay but access times stored in slaves on a primary bus unit should not be modified.

Persons skilled in the art will recognize that a more sophisticated transceiver can control transmissions to and from primary bus units. An additional control line, TrncvrRW can be bused to all devices on the transceiver bus, using that line in conjunction with the AddrValid line to indicate to all devices on the transceiv r bus that the information on the data lines is: 1) a request packet, 2) valid data to a slave, 3) valid data from a

slave, or 4) invalid data (or idle bus). Using this extra control line obviates the need for the transceivers to keep track of when data needs to be forwarded from its primary bus to the transceiver bus - all transceivers send all data from their primary bus to the transceiver bus whenever the control signal indicates condition 2) above. In a preferred implementation of this invention, if AddrValid and TrncvrRW are both low, there is no bus activity and the transceivers should remain in an idle state. A controller sending a request packet will drive AddrValid high, indicating to all devices on the transceiver bus that a request packet is being sent which each transceiver should forward to its primary bus unit. Each controller seeking to write to a slave should drive both AddrValid and TrncvrRW high, indicating valid data for a slave is present on the data lines. Each transceiver device will then transmit all data from the transceiver bus lines to each primary bus unit. Any controller expecting to receive information from a slave should also drive the TrncvrRW line high, but not drive AddrValid, thereby indicating to each transceiver to transmit any data coming from any slave on its primary local bus to the transceiver bus. A still more sophisticated transceiver would recognize signals addressed to or coming from its primary bus unit and transmit signals only at requested times.

An example of the physical mounting of the transceivers is shown in Figure 9. One important feature of this physical

arrangement is to integrate the bus of each transceiver 19 with the original bus of DRAMs or other devices 15, 16, 17 on the primary bus unit 66. The transceivers 19 have pins on two sides, and are preferably mounted flat on the primary bus unit with a first set of pins connected to primary bus 18. A second set of transceiver pins 20, preferably orthogonal to the first set of pins, are oriented to allow the transceiver 19 to be attached to the transceiver bus 65 in much the same way as the DRAMs were attached to the primary bus unit. The transceiver bus can be generally planar and in a different plane, preferably orthogonal to the plane of each primary bus unit. The transceiver bus can also be generally circular with primary bus units mounted perpendicular and tangential to the transceiver bus.

Using this two level scheme allows one to easily build a system that contains over 500 slaves (16 buses of 32 DRAMs each). Persons skilled in the art can modify the device ID scheme described above to accommodate more than 256 devices, for example by using a longer device ID or by using additional registers to hold some of the device ID. This scheme can be extended in yet a third dimension to make a second-order transceiver bus, connecting multiple transceiver buses by aligning transceiver bus units parallel to and on top of each other and busing corresponding signal lines through a suitable transceiver. Using such a second-order transceiver bus, one

could connect many thousands of slave devices into what is effectively a single bus.

Device Interface

The device interface to the high-speed bus can be divided into three main parts. The first part is the electrical interface. This part includes the input receivers, bus drivers and clock generation circuitry. The second part contains the address comparison circuitry and timing registers. This part takes the input request packet and determines if the request is for this device, and if it is, starts the internal access and delivers the data to the pins at the correct time. The final part, specifically for memory devices such as DRAMs, is the DRAM column access path. This part needs to provide bandwidth into and out of the DRAM sense amps greater than the bandwidth provided by conventional DRAMs. The implementation of the electrical interface and DRAM column access path are described in more detail in the following sections. Persons skilled in the art recognize how to modify prior-art address comparison circuitry and prior-art register circuitry in order to practice the present invention.

Electrical Interface - Input/Output Circuitry

A block diagram of the preferred input/output circuit for address/data/control lines is shown in Figure 10. This

circuitry is particularly well-suited for use in DRAM devices but it can be used or modified by one skilled in the art for use in other devices connected to the bus of this invention. It consists of a set of input receivers 71, 72 and output driver 76 connected to input/output line 69 and pad 75 and circuitry to use the internal clock 73 and internal clock complement 74 to drive the input interface. The clocked input receivers take advantage of the synchronous nature of the bus. To further reduce the performance requirements for device input receivers, each device pin, and thus each bus line, is connected to two clocked receivers, one to sample the even cycle inputs, the other to sample the odd cycle inputs. By thus de-multiplexing the input 70 at the pin, each clocked amplifier is given a full 2 ns cycle to amplify the bus low-voltage-swing signal into a full value CMOS logic signal. Persons skilled in the art will recognize that additional clocked input receivers can be used within the teachings of this invention. For example, four input receivers could be connected to each device pin and clocked by a modified internal device clock to transfer sequential bits from the bus to internal device circuits, allowing still higher external bus speeds or still longer settling times to amplify the bus low-voltage-swing signal into a full value CMOS logic signal.

The output drivers are quite simple, and consist of a single NMOS pulldown transistor 76. This transistor is sized so that under worst case conditions it can still sink the 50 mA

required by the bus. For 0.8 micron CMOS t chnology, the transistor will need to be about 200 microns long. Overall bus performance can be improved by using feedback techniques to control output transistor current so that the current through the d vice is roughly 50 mA under all operating conditions, although this is not absolutely necessary for proper bus operation. An example of one of many methods known to persons skilled in the art for using feedback techniques to control current is described in Hans Schumacher, et al., "CMOS Subnanosecond True-ECL Output Buffer," *J. Solid State Circuits*, Vol. 25 (1), pp. 150-154 (Feb. 1990). Controlling this current improves performance and reduces power dissipation. This output driver which can be operated at 500 MHz, can in turn be controlled by a suitable multiplexer with two or more (preferably four) inputs connected to other internal chip circuitry, all of which can be designed according to well known prior art.

The input receivers of every slave must be able to operate during every cycle to determine whether the signal on the bus is a valid request packet. This requirement leads to a number of constraints on the input circuitry. In addition to requiring small acquisition and resolution delays, the circuits must take little or no DC power, little AC power and inject very little current back into the input or reference lines. The standard clocked DRAM sense amp shown in Figure 11 satisfies all these requirements exc pt the need for low input currents. When

this sense amp goes from sense to sample, the capacitance of the internal nodes 83 and 84 in Figure11 is discharged through the reference line 68 and input 69, respectively. This particular current is small, but the sum of such currents from all the inputs into the reference lines summed over all devices can be reasonably large.

The fact that the sign of the current depends upon on the previous received data makes matters worse. One way to solve this problem is to divide the sample period into two phases. During the first phase, the inputs are shorted to a buffered version of the reference level (which may have an offset). During the second phase, the inputs are connected to the true inputs. This scheme does not remove the input current completely, since the input must still charge nodes 83 and 84 from the reference value to the current input value, but it does reduce the total charge required by about a factor of 10 (requiring only a 0.25V change rather than a 2.5V change). Persons skilled in the art will recognize that many other methods can be used to provide a clocked amplifier that will operate on very low input currents.

One important part of the input/output circuitry generates an internal device clock based on early and late bus clocks. Controlling clock skew (the difference in clock timing between devices) is important in a system running with 2 ns cycles, thus the internal device clock is generated so the input

sampler and the output driver operate as close in time as possible to midway between the two bus clocks.

A block diagram of the internal device clock generating circuit is shown in Figure 12 and the corresponding timing diagram in Figure 13. The basic idea behind this circuit is relatively simple. A DC amplifier 102 is used to convert the small-swing bus clock into a full-swing CMOS signal. This signal is then fed into a variable delay line 103. The output of delay line 103 feeds three additional delay lines: 104 having a fixed delay; 105 having the same fixed delay plus a second variable delay; and 106 having the same fixed delay plus one half of the s cond variable delay. The outputs 107, 108 of the delay lines 104 and 105 drive clocked input receivers 101 and 111 connected to early and late bus clock inputs 100 and 110, respectively. These input receivers 101 and 111 have the same design as the receivers described above and shown in Fig. 11. Variable delay lines 103 and 105 are adjusted via feedback lines 116, 115 so that input receivers 101 and 111 sample the bus clocks just as they transition. Delay lines 103 and 105 are adjusted so that the falling edge 120 of output 107 precedes the falling edge 121 of the early bus clock, Clock1 53, by an amount of time 128 equal to the delay in input sampler 101. Delay line 108 is adjusted in the same way so that falling edge 122 precedes the falling edge 123 of late bus clock, Clock2 54, by the delay 128 in input sampler 111.

Since the outputs 107 and 108 are synchronized with the two bus clocks and the output 73 of the last delay line 106 is midway between outputs 107 and 108, that is, output 73 follows output 107 by the same amount of time 129 that output 73 precedes output 108, output 73 provides an internal device clock midway between the bus clocks. The falling edge 124 of internal device clock 73 precedes the time of actual input sampling 125 by one sampler delay. Note that this circuit organization automatically balances the delay in substantially all device input receivers 71 and 72 (Fig. 10), since outputs 107 and 108 are adjusted so the bus clocks are sampled by input receivers 101 and 111 just as the bus clocks transition.

In the preferred embodiment, two sets of these delay lines are used, one to generate the true value of the internal device clock 73, and the other to generate the complement 74 without adding any inverter delay. The dual circuit allows generation of truly complementary clocks, with extremely small skew. The complement internal device clock is used to clock the 'even' input receivers to sample at time 127, while the true internal device clock is used to clock the 'odd' input receivers to sample at time 125. The true and complement internal device clocks are also used to select which data is driven to the output drivers. The gate delay between the internal device clock and output circuits driving the bus is slightly greater than the corresponding delay for the input circuits, which means that the

new data always will be driven on the bus slightly after the old data has been sampled.

DRAM Column Access Modification

A block diagram of a conventional 4 MBit DRAM 130 is shown in Figure 15. The DRAM memory array is divided into a number of subarrays 150-157, for example, 8. Each subarray is divided into arrays 148, 149 of memory cells. Row address selection is performed by decoders 146. A column decoder 147A, 147B, including column sense amps on either side of the decoder, runs through the core of each subarray. These column sense amps can be set to precharge or latch the most-recently stored value, as described in detail above. Internal I/O lines connect each set of sense-amps, as gated by corresponding column decoders, to input and output circuitry connected ultimately to the device pins. These internal I/O lines are used to drive the data from the selected bit lines to the data pins (some of pins 131-145), or to take the data from the pins and write the selected bit lines. Such a column access path organized by prior art constraints does not have sufficient bandwidth to interface with a high speed bus. The method of this invention does not require changing the overall method used for column access, but does change implementation details. Many of these details have been implemented selectively in certain fast memory devices, but never in conjunction with the bus architecture of this invention.

Running the internal I/O lines in the conventional way at high bus cycle rates is not possible. In the preferred method, several (preferably 4) bytes are read or written during each cycle and the column access path is modified to run at a lower rate (the inverse of the number of bytes accessed per cycle, preferably 1/4 of the bus cycle rate). Three different techniques are used to provide the additional internal I/O lines required and to supply data to memory cells at this rate. First, the number of I/O bit lines in each subarray running through the column decoder 147 is increased, for example, to 16, eight for each of the two columns of column sense amps and the column decoder selects one set of columns from the "top" half 148 of subarray 150 and one set of columns from the "bottom" half 149 during each cycle, where the column decoder selects one column sense amp per I/O bit line. Second, each column I/O line is divided into two halves, carrying data independently over separate internal I/O lines from the left half 147A and right half 147B of each subarray (dividing each subarray into quadrants) and the column decoder selects sense amps from each right and left half of the subarray, doubling the number of bits available at each cycle. Thus each column decode selection turns on *n* column sense amps, where *n* equals four (top left and right, bottom left and right quadrants) times the number of I/O lines in the bus to each subarray quadrant (8 lines each x 4=32 lines in the preferred implementation). Finally, during each RAS cycle,

two different subarrays, e.g. 157 and 153, are accessed. This doubles again the available number of I/O lines containing data. Taken together, these changes increase the internal I/O bandwidth by at least a factor of 8. Four internal buses are used to route these internal I/O lines. Increasing the number of I/O lines and then splitting them in the middle greatly reduces the capacitance of each internal I/O line which in turn reduces the column access time, increasing the column access bandwidth even further.

The multiple, gated input receivers described above allow high speed input from the device pins onto the internal I/O lines and ultimately into memory. The multiplexed output driver described above is used to keep up with the data flow available using these techniques. Control means are provided to select whether information at the device pins should be treated as an address, and therefore to be decoded, or input or output data to be driven onto or read from the internal I/O lines.

Each subarray can access 32 bits per cycle, 16 bits from the left subarray and 16 from the right subarray. With 8 I/O lines per sense-amplifier column and accessing two subarrays at a time, the DRAM can provide 64 bits per cycle. This extra I/O bandwidth is not needed for reads (and is probably not used), but may be needed for writes. Availability of write bandwidth is a more difficult problem than read bandwidth because over-writing a value in a sense-amplifier may be a slow operation, depending on how the sense amplifi r is connected to the bit line. The

extra set of internal I/O lines provides some bandwidth margin for write operations.

Persons skilled in the art will recognize that many variations of the teachings of this invention can be practiced that still fall within the claims of this invention which follow.